Assignment 1

Que1: Implement a program(s) to list the first 50 fibonacci numbers preferably in C/C++ in the following manner

1. **Implementation**
    - a. Recursion

```
function fibo_recursion(length) {
        if (length <= 1) return length;
        return fibo_main(length - 1) + fibo_main(length - 2);
    }
```

    - b. Loop

```
function fibo_loop(length) {
    if (length <= 0) return [];
    if (length === 1) return [0];

    let fiboNumbers = [0, 1];
    var second_priv = 0;
    var priv = 1;
    for (var i = 2; i < length; i++) {
        const temp = priv
        priv = priv + second_priv
        second_priv = temp
        fiboNumbers.push(priv)
    }
    return fiboNumbers
}
```

    - c. Recursion and Memoization

```
function fibo_recursionMemoization(length) {
        if (length <= 1) return length;
        if(memo[length]) return memo[length];
        memo[length] = fibo_main(length - 1) + fibo_main(length - 2);
        return memo[length];
    }
```

    - d. Loop with Memoization

```
function fibo_loopMemoization(length) {
    if (length <= 0) return [];
    if (length === 1) return [0];
    if (length === 2) return [0, 1];

    let fiboNumbers = [0, 1];
    for (var i = 2; i < length; i++) {
        const nextFib = fiboNumbers[i - 1] + fiboNumbers[i - 2];
        fiboNumbers.push(nextFib);
    }
    return fiboNumbers
}
```

2. **Performance Evaluation**

   The speedup factor was calculated as follows:

   $Speedup = BaselineTime/MethodTime$, where Baseline Time is time taken by Recursion(b)

   Baseline Time (Recursion(b)): 2460.99775

	Time (ms)	Speedup
Loop(a)	0.029666999999790278	82954.048269707
Recursion and Memoization(c)	0.1586250000000291	15514.56422379542
Loop and Memoization(d)	0.1586250000000291	49343.313283053256

Que2. Write a simple Matrix Multiplication program for a given NxN matrix in any two of your preferred Languages from the following listed buckets, where N is iterated through the set of values 64, 128, 256, 512 and 1024. N can either be hardcoded or specified as input. Consider two cases (a) Elements of matrix are of data type Integer and (b) Double In each case, (i.e. Bucket 1 for (a) and (b) + Bucket 2 for (a) and(b))

Bucket2: Python

- a. Report the output of the ‘time’ describing the system and CPU times.

  Execution ie, **System** times for matrix multiplication where N and Elements are as follows:

N	Elements are Integers	Elements are Doubles
64	0.2946035861968994 seconds	0.28389549255371094 seconds
128	2.397904396057129 seconds	2.1375110149383545 seconds
256	21.47019600868225 seconds	19.850048065185547 seconds
512	171.9491012096405 seconds	155.66996312141418 seconds
1024	1388.12477684021 seconds	1282.716549873352 seconds

CPU Time is real time that computer spends to execute the main code (here, matrix multiplication code) whereas, **System Time is** total time that computer take to run whole program it may, memory allocation, various other operations like input/output and variables assignation.

So, **CPU** takes less time than **System.**

- b. Using the 'language hooks' evaluate the execution time for the meat portions of the program and how much proportion is it w.r.t. total program execution time.

  Assumption: The total program execution time refers to the time taken for that particular value of N.

  for Intiger Matrix Multiplication

N	Total time	Meat portion	Proportion
64	0.24181664299999284 seconds	0.23953979999998865 seconds	99.05844239182319%
128	1.9395357539999623 seconds	1.9369777709999888 seconds	99.86811364551033%
256	16.431510711999977 seconds	16.429971000000023 seconds	99.9906295165007%
512	140.94247064299998 seconds	140.936826902 seconds	99.9959957130209%
1024	1147.6208251459998 seconds	1147.5975050230004 seconds	99.99796795923459%

for Double Matrix Multiplication

N	Total time	Meat portion	Proportion
64	0.41083479600001 74 seconds	0.40702503900001 83 seconds	99.0726790824215 2%
128	2.66562191999992 14 seconds	2.66448374200012 95 seconds	99.9573015966273 3%
256	16.8482564649998 5 seconds	16.8458691959999 67 seconds	99.9858307653089 2%
512	143.1886671510001 2 seconds	143.180762512 seconds	99.9944795638108 8%
1024	1147.620825145999 8 seconds	1147.597505023000 4 seconds	99.9979679592345 9%

Bucket1: C

- a. Report the output of the ‘time’ describing the system and CPU times.

  Execution ie, **System** times for matrix multiplication where N and Elements are as follows:

N	Elements are Integers	Elements are Doubles
64	0.002528 seconds	2.753615 seconds
128	0.013937 seconds	0.007373 seconds
256	0.067842 seconds	0.059308 seconds
512	0.331150 seconds	0.479018 seconds
1024	2.75615 seconds	4.844258 seconds

  CPU Time is real time that computer spends to execute the main code (here, matrix multiplication code) whereas, **System Time is** total time that computer take to run whole program it may, memory allocation, various other operations like input/output and variables assignation.

  So, **CPU** takes less time than **System.**

- b. Using the ‘language hooks’ evaluate the execution time for the meat portions of the program and how much proportion is it w.r.t. total program execution time.

  Assumption: The total program execution time refers to the time taken for that particular value of N.

  for Intiger Matrix Multiplication

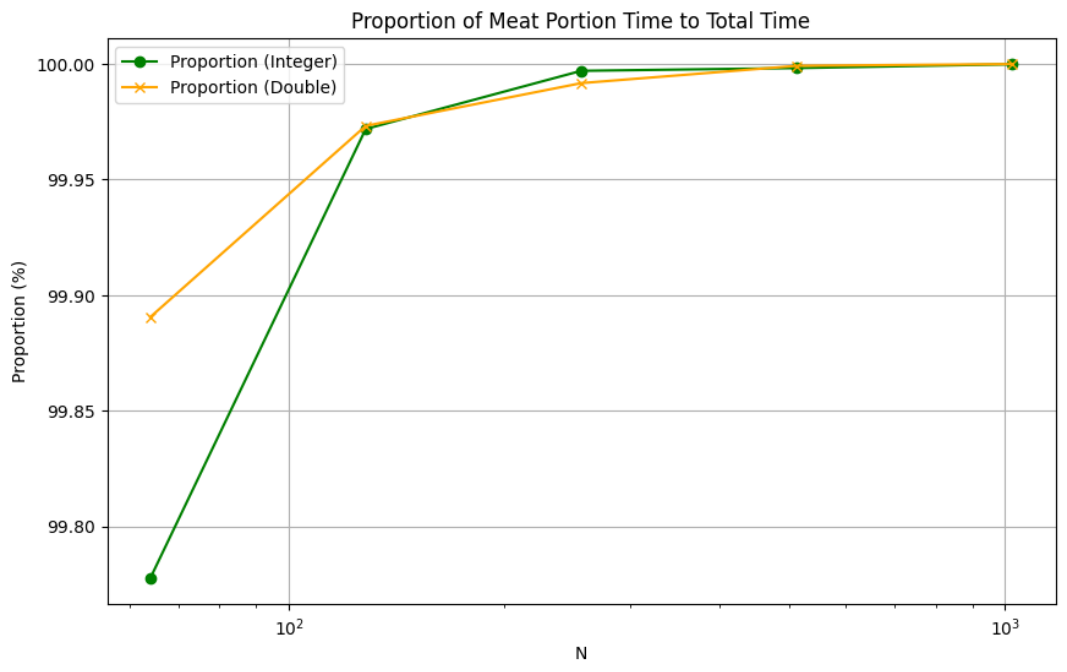
N	Total time	Meat portion	Proportion
64	0.002247 seconds	0.002242 seconds	99.777481%
128	0.010658 seconds	0.010655 seconds	99.971852%
256	0.066923 seconds	0.066921 seconds	99.997011%
512	0.333853 seconds	0.333847 seconds	99.998203%
1024	2.741945 seconds	2.741941 seconds	99.999854%

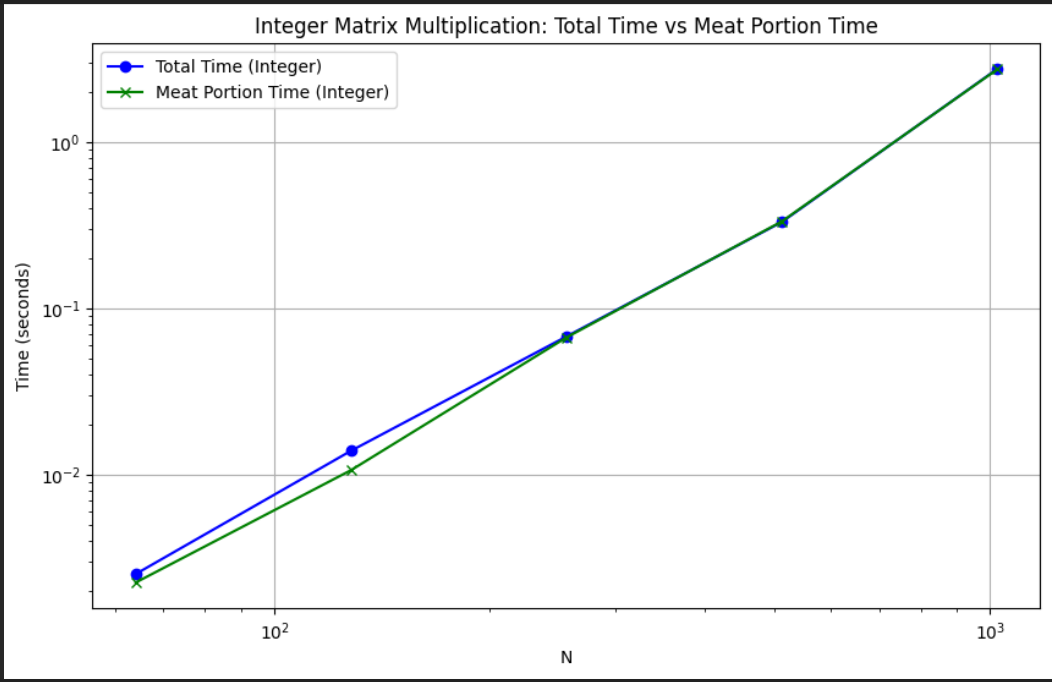
for Double Matrix Multiplication

N	Total time	Meat portion	Proportion
64	0.000913 seconds	0.000912 seconds	99.890471%
128	0.007464 seconds	0.007462 seconds	99.973205%
256	0.060135 seconds	0.060130 seconds	99.991685%
512	0.483210 seconds	0.483206 seconds	99.999172%
1024	4.721904 seconds	4.721901 seconds	99.999936%

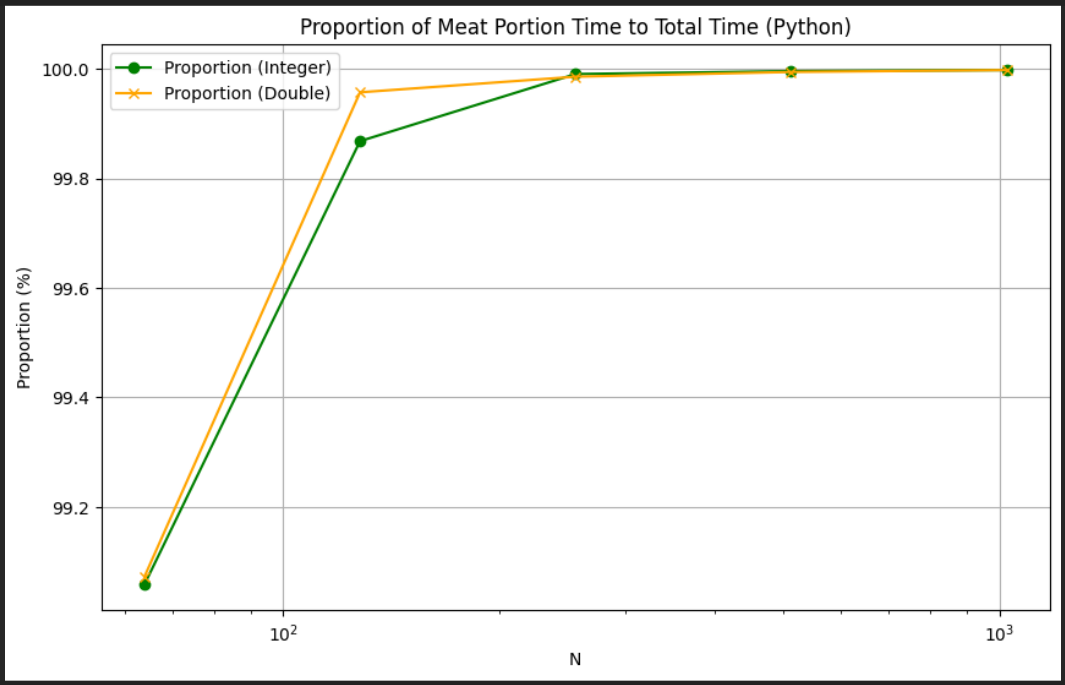
c.

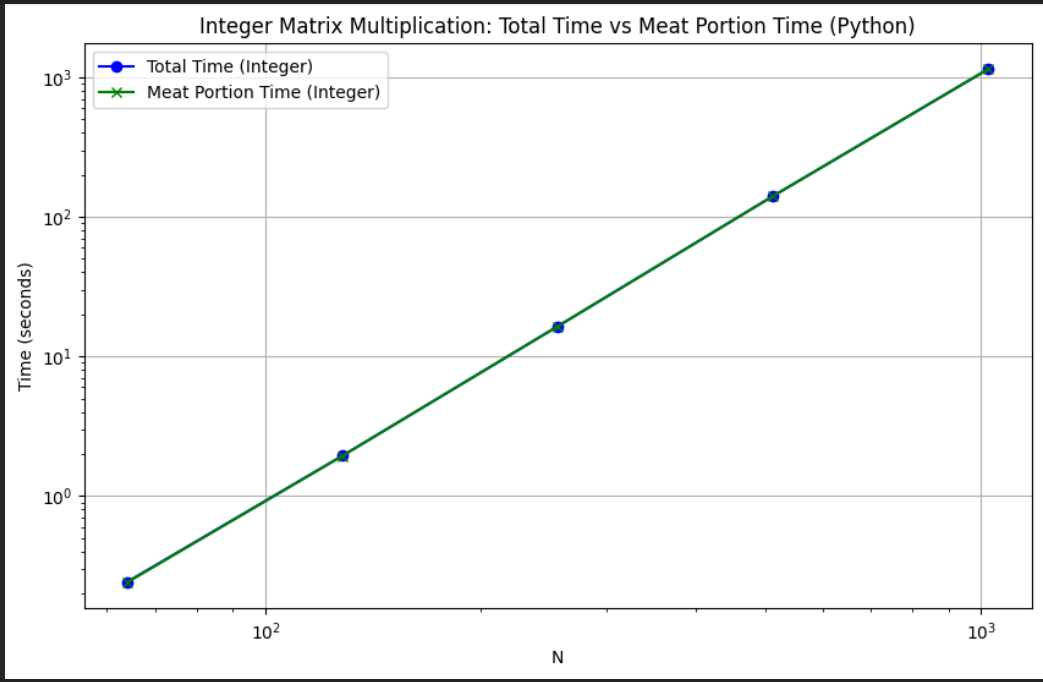
C Plot





Python Plot





Observations were as expected C take lesser time when compared to python. On trend can be seen is when we compare the Meat portion and Total time we will see that the Proportion is purely based on Meat part as the other time is more or less same through out. and as N increase the complexity of code and run time too increases, also as we move on incresing N the Proportion starts steping down or make make a constant curve

GitHub Code: [ES-215-Computer-Organization-and-Architecture/Assignment1 at main · vipulSP2108/ES-215-Computer-Organization-and-Architecture · GitHub]